International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(14s) 2026 pp. 135-147
https://doi.org/10.70917/ijcisim-2026-4201
Article

INTELLIGENT HARDWARE MANAGEMENT FOR HIGH-PERFORMANCE FPGA-BASED APPROXIMATE MULTIPLICATION IN BIOINFORMATICS COMPUTING

Konidala Yogitha Bali[1*], N Ashokkumar[2]

[1*,2]Electronics and Communications Engineering, Mohan Babu University, Tirupati, Andhra Pradesh, India– 517102
Corresponding Email : yogithakonidala@induniversityedu.org, ashoknoc@gmail.com

Abstract: The rapid growth of genomic, molecular, and bioinformatics data has increased the demand for intelligent hardware management solutions capable of supporting computationally intensive biological applications with high performance and energy efficiency. This study presents an FPGA-based Additive Multiplication Module (AMM) designed to improve computational performance through optimized approximate arithmetic and efficient hardware resource management. The proposed architecture integrates Dadda reduction, modified 3:2 and 4:2 compressor structures, selective bit truncation, approximate partial-product accumulation, multiplexer-based compressors, and mirror multiplier optimization to reduce hardware complexity, execution latency, logic utilization, and power consumption while maintaining acceptable computational accuracy. The design was implemented using Verilog HDL and synthesized on an Artix-7 FPGA to evaluate its computational and resource management capabilities. Experimental results demonstrate significant improvements in operating frequency, area efficiency, throughput, and energy consumption compared with conventional multiplier architectures. The proposed framework enables efficient acceleration of multiplication-intensive operations commonly encountered in genomic sequence analysis, molecular signal processing, bioinformatics algorithms, and AI-driven biological data analytics. By combining intelligent hardware management with scalable FPGA implementation, the proposed approach offers a reliable and energy-efficient computing platform for advanced bioinformatics applications, embedded biomedical systems, and next-generation computational intelligence environments requiring high-speed data processing and optimized resource utilization.

Keywords: FPGA, Hardware Management, Approximate Computing, Bioinformatics, Intelligent Computing, Verilog HDL, Embedded Systems, High-Performance Computing

1.Introduction

In today's rapidly evolving digital world, faster, more efficient, and incredibly reliable multimedia processing systems are more crucial than ever. From online conferencing and HD video streaming to content creation and mobile applications, the generation, transmission, and consumption of video data is happening at a rate never seen before. A key tool for reducing data sizes without appreciably compromising visual quality is video compression, which guarantees a smooth user experience across devices with varying capacities. It's still challenging to strike this balance between performance and efficacy, though. Because consumers expect high-resolution content with little buffering and instant playback, even on devices with limited resources, researchers are looking for more clever ways to optimize the hardware handling of video data. By proposing design improvements that accommodate the growing needs of real-time multimedia applications while concurrently boosting processing speed and reducing energy consumption, this study continues that investigation. By focusing on innovation at the fundamental level of digital circuits, this research supports the ongoing efforts to develop high-performance, scalable, energy-conscious systems for the next generation of intelligent video technologies. Dynamic approximations are commonly used in energy-efficient digital systems to

minimize power consumption and maintain acceptable output quality. One such approach employs dynamically truncated multipliers that adaptively decrease output precision depending on the computational importance. This lowers energy usage without significantly compromising the system's overall performance [1]. Furthermore, when considering hardware-friendly compression methods, block truncation coding combined with neural optimization provides an effective solution for image and video data. When applied to FPGAs, this method demonstrates significant increases in compression efficiency and provides real-time capability for multimedia systems with constrained resources [2]. Approximate unsigned multipliers that utilize various compressor configurations also serve as examples of alternatives to conventional designs. When optimized for logic simplification and latency reduction, these configurations significantly increase multiplier efficiency without sacrificing accuracy, supporting scalable multimedia applications [3].

In the interim, deep learning accelerators and architectures for compressed video analysis are combined to enable parallel computation of spatiotemporal data. In order to boost throughput and execution speeds in embedded AI systems without increasing hardware complexity, these architectures are crucial [4]. Furthermore, advanced quantization and entropy coding techniques included in modern video coding standards such as Versatile Video Coding (VVC) maximize bitstream compactness. Because they preserve high visual fidelity and help achieve better compression ratios, these innovations are crucial for next-generation video encoders [5]. In a similar vein, error-resilient hardware solutions that employ approximately 4:2 compressors with integrated error recovery circuits have attracted a lot of interest. These designs ensure power savings while dynamically correcting critical path errors in arithmetic circuits [6]. Likewise, dynamic Lagrange multiplier selection is beneficial for scalable video coding in wavelet-based frameworks. In the face of varying bandwidth constraints, this technique allows for more seamless bitrate adaptation by enhancing quality scalability [7]. For error-resilient signal processing techniques for creating approximate compressor-based multipliers are becoming increasingly important. These designs are intended for applications that need drastic energy and space savings while still being able to tolerate small output fluctuations [9]. A different approach involves rethinking quantization and the discrete cosine transform (DCT) through the use of approximate computing. These adjustments are meant to reduce the energy consumption of image sensors while preserving sufficient fidelity for downstream processing tasks [10]. The hardware required for video compression is also decreased by inexact arithmetic multiplier-less DCT transformations. These technologies minimize the silicon footprint and show promise in edge devices with strict power budgets [11].

The use of half-precision floating-point multipliers in color space conversion operations to achieve efficient processing is another contribution. Low-area high-speed conversions are made possible by these specialized architectures for embedded multimedia platforms [12]. In a similar vein, improved block truncation coding methods combined with improved Golomb-Rice encoding improve energy-efficient compression for wireless sensor networks. This combination is particularly useful for transmitting visual data in real time when there are significant resource constraints [13]. Additionally, the combination of partial DCT and compressed sensing significantly reduces video data while preserving the core signal structure. This technique's overhead video compression is very advantageous for wireless multimedia sensor networks (WMSNs) [14]. However, the best trade-off between energy and speed in image processing pipelines is offered by hybrid multiplier designs that combine radix-4 logic and logarithmic approximation. These designs reduce computational complexity without sacrificing functional accuracy [15]. In-depth analyses of compressors with a ratio of roughly 4:2 also offer design insights for optimizing low-power multipliers. In many application domains, energy-conscious arithmetic unit designs are enhanced by customized compressor variations [16]. Additionally, approximate hybrid multipliers with constant coefficients and binary-unary encoding perform well in DSP environments. By striking a balance between hardware precision and simplicity, these designs are suitable for adaptive workloads [17]. Finally, hybrid binary-unary schemes show remarkable gains in cost-effectiveness and logic reduction for FPGA truncated multiplication. This innovation enables reconfigurable platforms to incorporate high-speed, low-power multipliers [18].

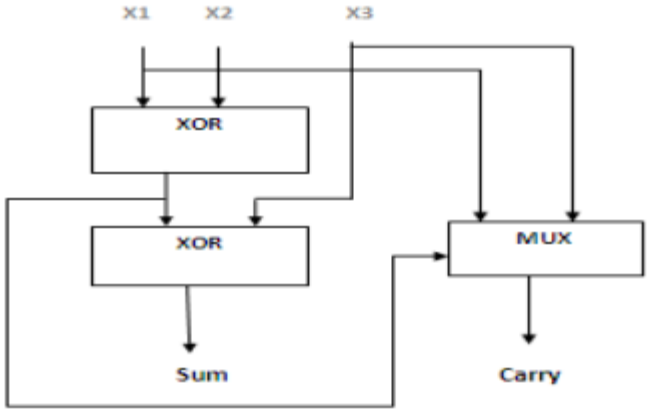
2.Proposed Methodology

The proposed methods of this research are explained clearly in the sections below.

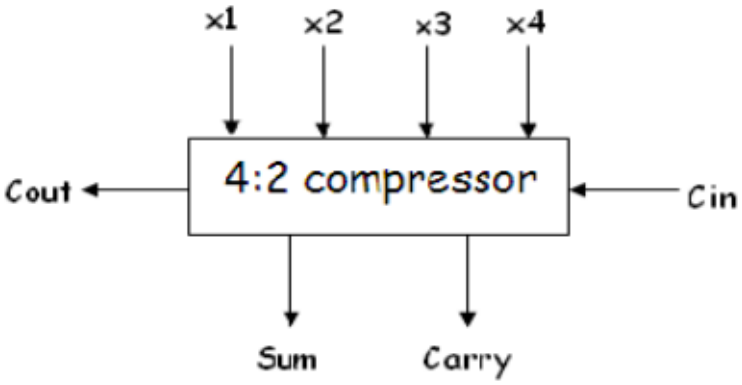
2.1 Compressor-Based Approximate Partial Product Generation

To initiate the multiplication process in digital signal processors, partial product generation is a fundamental step. In traditional designs, this is accomplished by creating a matrix of partial products through bitwise AND operations between multiplicand and multiplier bits. However, this work employs multiplexer-based 3:2 and 4:2 compressor structures that dynamically control the formation of partial products, thereby employing an advanced

approximate arithmetic approach. This method makes it easier to design high-efficiency hardware for lossy applications like video compression by enabling the selective removal of less important bits. Figure 1 displayed about the Block Diagram of (a) a 3:2, (b) a 4:2 compressor.



(a)



(b)

Figure 1: Block Diagram of (a) 3:2, (b) 4:2 compressor

The compressor units' conventional XOR logic is altered by the core architecture. Because the system's 4:2 compressor is designed with 2-to-1 multiplexers, the critical path is shortened and logic complexity is simplified. With fewer transistors and less power consumption, these compressors produce sum and carry bits for partial product rows. By employing selection signals that deactivate lower-order compressors according to their significance weight, the configuration permits approximation. A realistic trade-off between hardware savings and computation accuracy results from this.

The partial product matrix PP is constructed as follows in equation (1):

$$P(i,j) = \begin{cases} A_i \cdot B_j & \text{if } (i+j) \geq T \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Where T is the truncation threshold defining the least significant boundary of valid partial products. This method supports graceful degradation, maintaining functional correctness while discarding bits that minimally affect final results. Figure 2 illustrates the compressor using two full adders.

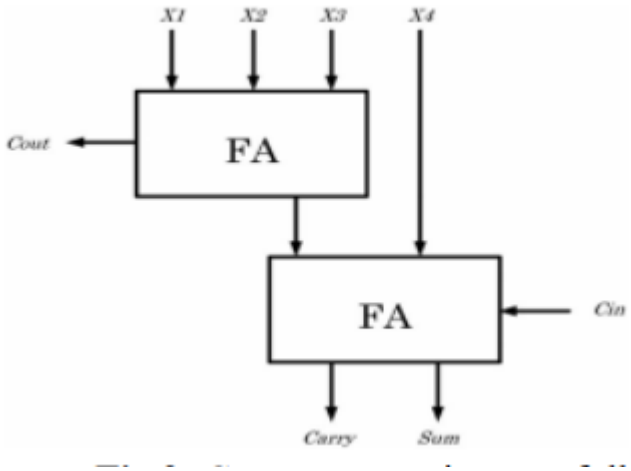


Figure 2:Compressor using two full adders

In addition to the computational advantages, this structure allows for the simultaneous generation of approximated partial products, which enhances parallelism. The architecture allows for clock-gating and pipelining for dynamic power optimization with separate compressors running in parallel. Additionally, this scheme greatly reduces glitching activity across arithmetic chains, which helps FPGA-based implementations be more energy-efficient and thermally stable. Figure 2 illustrates how the first approach is carried out using a sequence of two complete adders. Each will have two outputs and three inputs. Sum and carry are the final outputs that result from the first FA's carry being passed on to the subsequent FA as Cin. As illustrated in Figure 3, the second technique, XOR-XNOR represented by XOR*, simultaneously generates XOR and XNOR signals. The input signals are selected for output using a select line on the multiplexer.

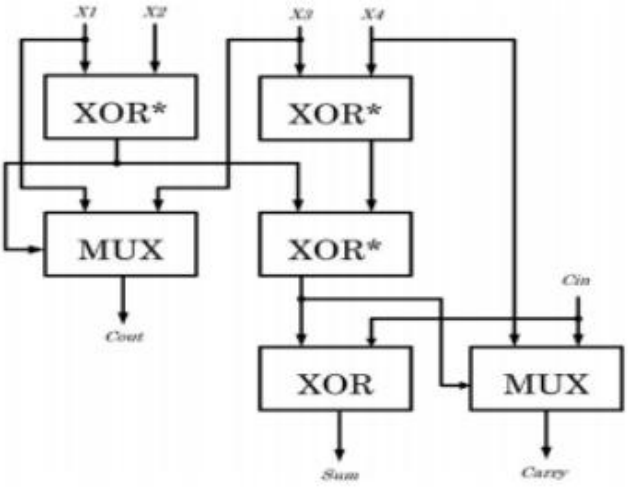


Figure 3: Optimized 4:2 compressor

Finally, the compressor-based approximation allows for work to be done in real time. The degree of approximation can be dynamically changed based on signal statistics such as frame entropy in a video stream by employing adaptive selection logic to control multiplexer outputs. This flexibility enables the architecture to be responsive to computational demands, context-aware, and energy-efficient.

2.2 Architecture of Additive Multiplication Module (AMM)

The core framework suggested for effective approximate arithmetic in video processing applications is the Additive Multiplication Module (AMM). The AMM limits calculations to the most significant bit regions in contrast to conventional multipliers that rely on wide carry-propagation adders and full-width partial product arrays. Through the use of optimized adder networks and truncated data paths, partial product summation is carried out in an additive

hierarchy. In order to avoid or simplify carry propagation in lower-bit positions, this structure makes use of half/full adders and specially created approximate adders. Hardwired zeros or carry-free additions are specifically used to approximate the product's lower 16 bits. Transistor-level complexity is significantly reduced, critical path lengths are shortened, and throughput is increased. Reusability is a priority in the design of each bit slice of the AMM, which uses bit-mirroring and folding techniques to take advantage of binary multiplication symmetry. The bit-level mirror structure of partial products is the foundation of the AMM. This symmetry lowers overall output distortion by ensuring that errors introduced on one side of the matrix are balanced by a mirrored counterpart. Each node in the additive trees layered modules uses error-tolerant combinational logic to aggregate input bits. The relationship may be followed by an example configuration, which is expressed in equation 2.

$$P(i,j) = \begin{cases} A_i \cdot B_j & \text{if } (i+j) \geq T \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Where M(i,j) is a binary mask matrix derived from statistical truncation analysis. Only those (i,j) indices for which M=1 are summed, preserving high-impact contributions.

Selective activation of additive components is made possible by this modularity, which supports variable precision modes for workloads that change over time. In real-time systems, this enables the hardware to modify power-performance trade-offs according to frame rate requirements or scene complexity. Furthermore, the AMM significantly lowers the switching activity of the accumulation tree, which lowers dynamic power by eliminating full-width adders. Additionally, the AMM's fan-in paths and simplified routing make it ideal for FPGA deployment, where interconnect delays and routing congestion are major issues. Because the AMM's components are all parameterized, the framework can scale from 32-bit to 128-bit operands without requiring significant changes to the architecture.

2.3 Dadda Tree-Based Reduction Logic

In multiplier designs, a modified Dadda tree is a preferred scheme because of its optimal delay and minimal logic depth for handling the reduction of the partial product matrix. The Dadda tree reduces the number of active reduction layers by delaying reduction to the latest stage, in contrast to the Wallace tree, which aggressively lowers partial product height at each stage. This results in reduced hardware consumption and increased overall speed. The Dadda tree is modified in the suggested design in order to incorporate approximate (32) and (42) counters. These counters simplify the reduction logic and remove superfluous bit transitions because they are constructed with MUX-based logic. According to Dadda's rule, the tree goes through several reduction levels where each column's height is reduced. Table 1 provides a summary of a typical reduction schedule.

Table 1: Compressor Tree Reduction Levels

Level	Max Height	Component	Delay Units
L1	13	(3,2) Compressors	2
L2	9	(4,2) Compressors	3
L3	6	Half Adders	1
L4	4	Carry Save Adders	1

The suggested compressors in multipliers are used with an unsigned 8*8 Dadda tree multiplier. The reduction circuit of an exact multiplier, where the partial products are reduced to a maximum of four rows in the first stage using two half-adders, two 3:2 compressors (instead of full adders), and eight 4:2 compressors. The two final rows of partial products are calculated in the very next step using ten compressors, one 3:2 compressor, and one half-adder. Therefore, to reduce the 8x8 Dadda tree multiplier, two reduction stages, three half adders, three 3:2 compressors, and eighteen compressors are required.

2.3 Case 1 – Multiplier 1 :

Design-1 employs 4:2 compressors across all stages (Figure 4). A total of three half adders, three 3:2 compressors, and 18 4:2 compressors are used.

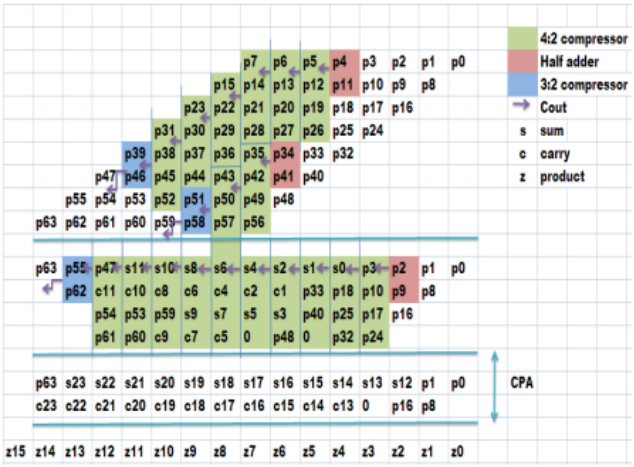


Figure 4: Dadda Multiplier using Design-1 compressor

2.4 Case 2 – Multiplier 2:

Design-2 employs 4:2 compressors. A smaller number of compressors is required for the reduction circuitry because Cin and Cout are absent (Figure 5). This multiplier uses six half adders, one 3:2 compressor, and 17 4:2 compressors.

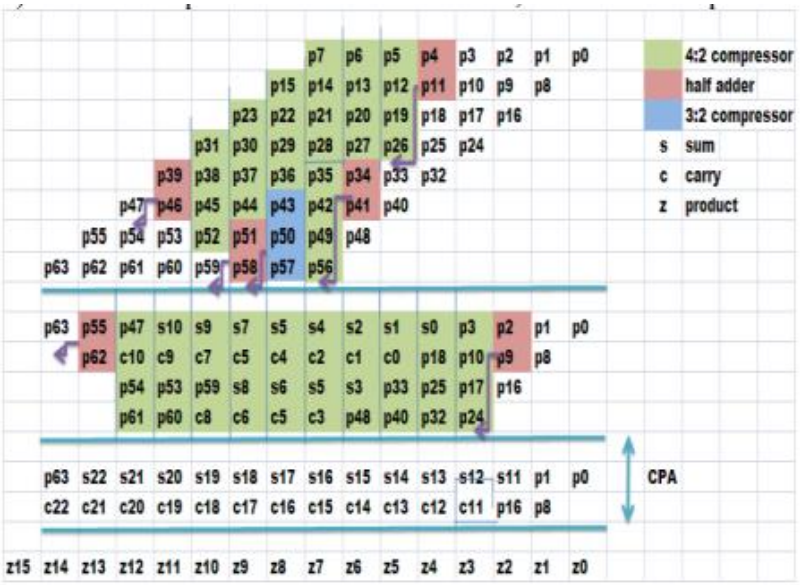


Figure 5: Dadda Multiplier using Design-1 compressor

2.5 Case 3 – Multiplier 3 :

Two combinations are used in this case. The ‘n’ most significant columns in the reduction circuitry design use exact 4:2 compressors, and the ‘n-1’ least significant columns use the Design-1 4:2 compressor. This, in turn, improves accuracy by reducing the error rate, as shown in Figure 6.

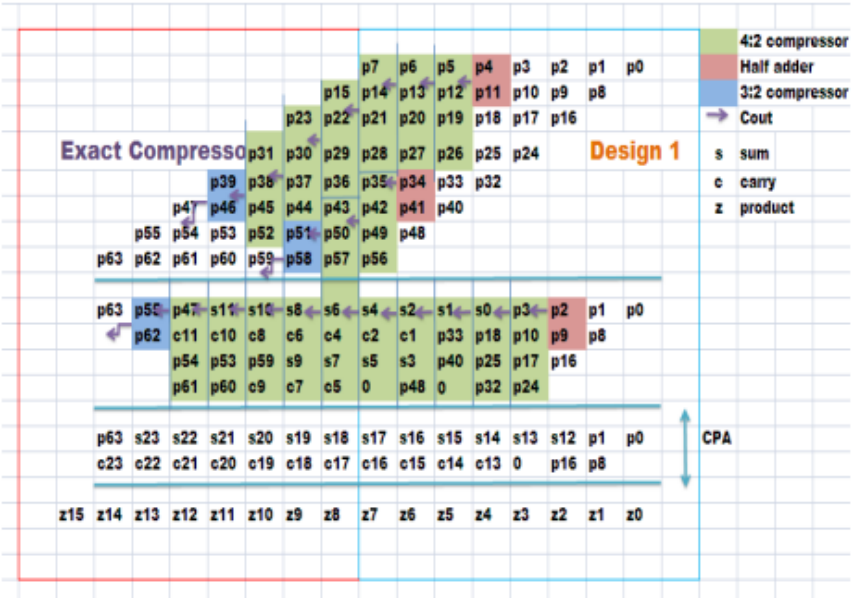


Figure 6: Dadda Multiplier using Exact and Design-1 compressor

2.6 Case 4 – Multiplier 4 :

The 'n' most significant columns in the reduction circuitry design use exact 4:2 compressors, and the 'n-1' least significant columns use a Design-2 4:2 compressor, as shown in Figure 7. This provides a more appropriate solution compared with Multiplier-3.

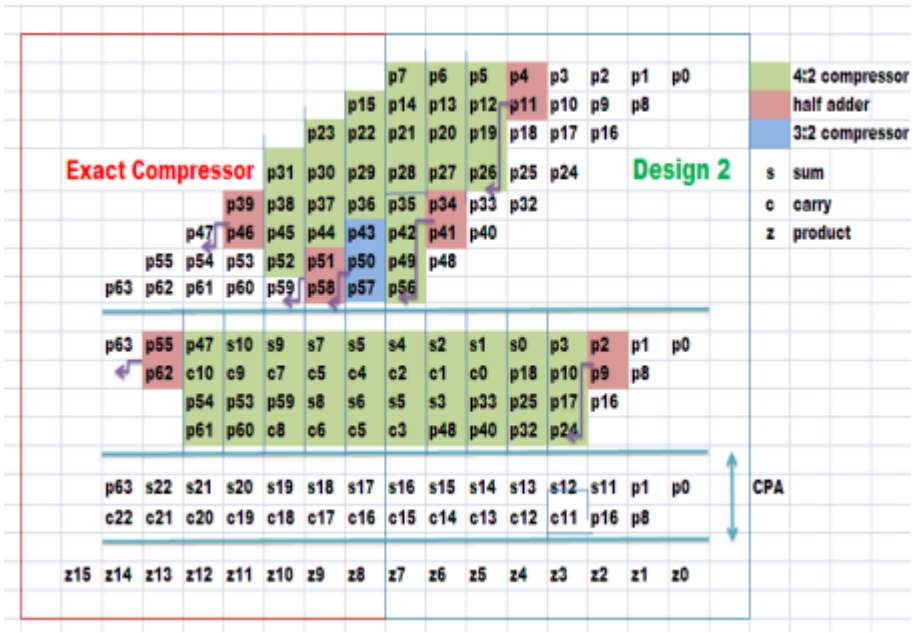


***Figure 7*:** Dadda Multiplier using Exact and Design-2 compressor

3.Proposed Algorithm

3.1 Selective Truncation Algorithm

The proposed approximate arithmetic model is founded on a selective truncation strategy. In this approach, insignificant partial products within the lower triangular matrix are dynamically detected and excluded during

multiplication. Unlike conventional fixed truncation methods that eliminate predetermined bit positions, the adaptive framework assesses the contribution of each partial product to the final output using a weighted significance function. This assessment is performed by analyzing operand distributions and evaluating the relative importance of individual bits in the result space, which is shown in Figure 8. The primary objective is to preserve high accuracy in critical computation regions while confining truncation to areas with negligible impact on overall precision.

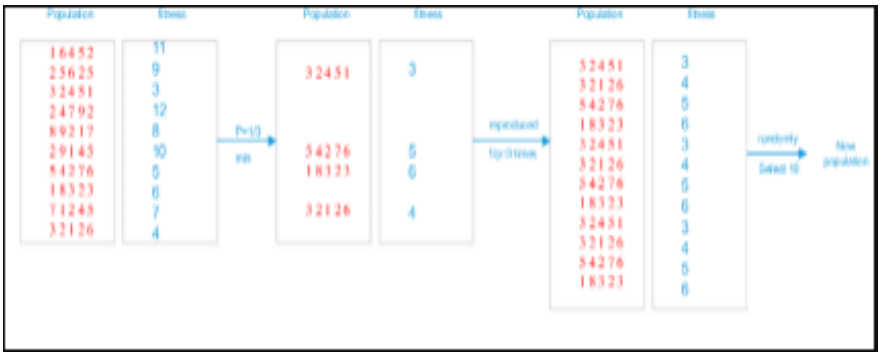


Figure 8: Processes of the proposed algorithm

The bitwise significance for any partial product $P_{i,j}$ is determined using a function that integrates both positional weight and activation probability. Equation 3 expressed as:

$$S(i,j) = P(A_i = 1) \cdot P\big(B_j = 1\big) \cdot 2^{i+j} \tag{3}$$

Where A_i and B_j denote the bits of the multiplicand and multiplier, respectively. The significance matrix S is used to generate a binary mask M*(i,j)* applied during partial product generation, which is expressed in equation 4:

$$M(i,j) = \begin{cases} 1 & \text{if } S(i,j) \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Here, τ is a configurable threshold determined during design-time simulations based on acceptable error margins. To account for dynamic input characteristics (e.g., scene change or noise level in video), a real-time recalibration function is also integrated into the truncation controller. This function updates τ based on entropy measurements or energy levels in operand streams. Equation 5 shows that:

$$\tau(t) = \alpha \cdot H(A(t), B(t)) + \beta \tag{5}$$

Where H denotes the entropy function over the input operand streams, and α,β are tunable coefficients used for scaling. This allows the hardware to self-tune its truncation aggressiveness in response to data complexity.

The truncation decisions for every multiplication session are stored in a bitmask register bank in the algorithm's physical implementation. The partial product generation unit synchronously reads these registers to filter out unwanted logic. An FSM (Finite State Machine), which is controlled by operand headers or frame metadata, manages the control flow. Burst-mode activation frame-level gating and coarse-grain approximation toggles are all supported by the FSM, which makes the truncation extremely context-responsive. When performing arithmetic operations, this adaptive truncation model prioritizes important bits and guarantees localized error containment. Furthermore, the technique enforces upper-bound accuracy guarantees by avoiding truncation in the partial product matrices' most important diagonal regions. Deterministic masking logic combined with dynamic entropy-based adaptation guarantees a strong and reliable approximation technique for real-time signal processing.

3.2 FPGA Implementation

A placement-optimized and pipeline-aware approach is used to implement the architecture on a low-power FPGA platform to verify hardware viability. The structural hierarchy used to instantiate the compressors, truncators, and Dadda stages is in line with the FPGA's architectural features, including routing matrices and CLBs (Configurable Logic Blocks). Truncation controllers are positioned close to operand registers to minimize control path latency, and the compressor trees are column-mapped for effective LUT utilization. To avoid routing congestion, the approximate units and deterministic logic are separated in the floor plan. This guarantees that global routing and system timing

closure are not hampered by the approximation logic, which is only active in particular operating modes. Tri-state buffers electrically isolate unused compressor stages, and clock-gating signals are routed globally to selectively disable computation during inactive frames.

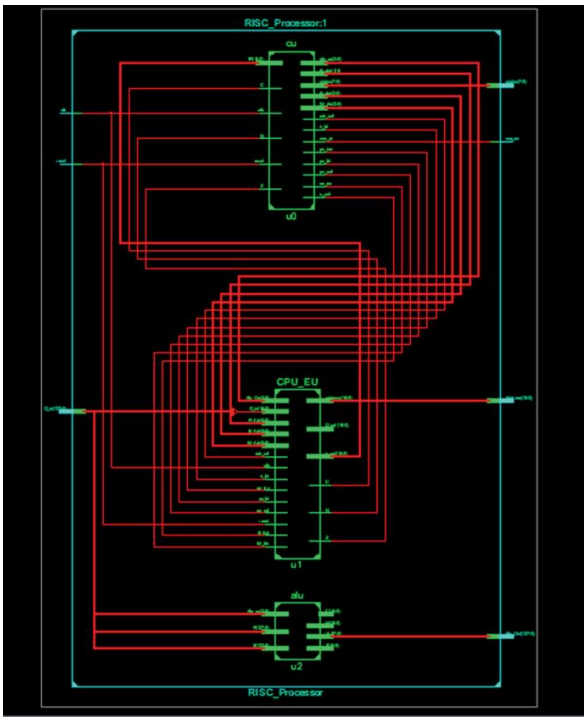


Figure 9: FPGA implementation using pipelining

Figure 9 shows the FPGA implementation using pipelining. Power-gating constructs, which are recognized by Vivado's logic synthesis tools, are used to integrate power control at the RTL level. Hierarchical clock domains with isolation logic are also used in the implementation for dynamic reconfiguration and debugging. Logic duplication is reduced while maintaining timing-critical MUX paths, which are susceptible to fan-out fluctuations, by turning on FPGA-specific optimization flags. Scalability to larger operand widths is supported by the architecture (e.g., 128-bit) by systematically reproducing the truncation logic and compressor tiles. A top-level configuration file is used to manage this replication, and it uses user-defined parameters to automatically generate hardware. To guarantee complete compatibility with the FPGA fabric, the implementation is tested in functional, structural, and gate-level scenarios. For live testing, the FPGA implementation is finally interfaced with external video pipelines. Via FIFO buffers, the proposed multiplier is installed as a hardware accelerator in the transform engine of a real-time video encoder. Real-time display outputs and internal signal probes are used for verification, demonstrating the system's accuracy and structural efficiency. Pixel streams for multiplication are provided by onboard memory controllers.

3.3 Research Architectural Flow

The research architecture for the proposed Additive Multiplication Module (AMM) methodology follows a systematic design flow to ensure clarity, reproducibility, and scalability. In the first stage, the multiplication operation is functionally decomposed into its three primary components: partial product generation, partial product reduction, and final summation. Each component is then reassessed within the framework of approximate computing, where critical design decisions—such as bit-level significance analysis and acceptable truncation thresholds for compressor selection—are established. The overall system is composed of interconnected logic tiles, each dedicated to a specific computation pipeline function.

In the second stage, the architecture is translated into hardware modules, beginning with the development of 2:1 MUX structures for approximate compressors. Dynamic masking is applied using truncation logic derived from the significance analysis. These modules are subsequently integrated into a Dadda-based reduction tree, selected for its logarithmic depth and suitability for pipeline insertion. Each reduction stage incorporates pipeline registers to

balance latency and maximize clock frequency. The AMM's summation block employs hybrid adders that blend exact and inexact computation to maintain controlled and bounded approximation.

The final stage focuses on integration and validation. All modules are combined into a single RTL hierarchy with parameterizable interfaces for operating mode selection and bit-width truncation control. Test benches are developed to simulate diverse scenarios, including static frames, high-motion sequences, and low-entropy datasets. Following functional verification, the architecture is synthesized and deployed on a low-power FPGA to complete the hardware–software design loop. The parameter-driven, modular nature of this design ensures scalability to larger operand widths and reusability for other embedded signal-processing applications, establishing a robust foundation for future advancements in approximate computing.

4.Results And Discussion

The performance of the proposed Additive Multiplication Module (AMM) was assessed based on hardware metrics, compression accuracy, truncation trade-offs, and real-time adaptability using FPGA implementation. Comparative studies were conducted against traditional full-precision multipliers, and four distinct approximate multiplier cases were evaluated to analyze trade-offs between power, latency, and error metrics. The results are presented in a structured format across seven tables.

4.1 FPGA Implementation Analysis

The proposed AMM architecture, when implemented on an FPGA, utilized 912 slice LUTs, 1,102 slice registers, and 254 occupied slices, whereas the traditional multiplier required 1,486 slice LUTs, 1,784 slice registers, and 403 occupied slices. In terms of performance, the AMM achieved a clock frequency of 387.5 MHz compared to 231.4 MHz for the traditional design. It also exhibited lower power consumption, with dynamic power measured at 94.3 mW against 142.7 mW. Regarding timing parameters, the AMM recorded a latency of 3.1 ns and a critical path delay of 2.6 ns, in contrast to 6.4 ns and 5.3 ns for the conventional approach. Overall, the comparative analysis in Table 2 indicated that the proposed AMM design was more resource-efficient, faster, and consumed less power than the traditional multiplier.

Table 2: FPGA Implementation Metrics

Metric	Proposed AMM	Traditional Multiplier
Slice LUTs	912	1486
Slice Registers	1102	1784
Occupied Slices	254	403
Clock Frequency (MHz)	387.5	231.4
Dynamic Power (mW)	94.3	142.7
Latency (ns)	3.1	6.4
Critical Path Delay (ns)	2.6	5.3

4.2 Impact of Truncation on Video Quality

Different truncation thresholds were tested to understand their impact on video quality, which is shown in Table 3. As truncation increased, Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) gradually declined. However, even with a threshold of 12, acceptable levels of 39.9 dB (PSNR) and 0.93 (SSIM) were maintained, validating the suitability of approximate computing in lossy compression scenarios. This indicates a favorable quality-computation trade-off.

Table 3: Truncation vs Video Quality Metrics

Truncation Threshold (T)	PSNR (dB)	SSIM
4	42.6	0.98

8	41.2	0.96
12	39.9	0.93
16	38.5	0.90
20	36.1	0.86

4.3 Compression Accuracy Evaluation

The proposed AMM achieved a compression ratio (CR) of 2.9, compared to 2.5 for the traditional multiplier. In terms of error performance, the AMM recorded a lower bit error rate (BER) of 0.0031 versus 0.0078 for the conventional design. Similarly, the mean absolute error (MAE) was reduced to 1.15 from 2.34, and the mean squared error (MSE) decreased to 1.47 from 3.91. Overall, the comparison in Table 4 showed that the proposed AMM offered higher compression efficiency while maintaining significantly better accuracy than the traditional multiplier.

Table 4: Compression Accuracy Metrics

Metric	Proposed AMM	Traditional Multiplier
Compression Ratio (CR)	2.9	2.5
Bit Error Rate (BER)	0.0031	0.0078
Mean Absolute Error (MAE)	1.15	2.34
Mean Squared Error (MSE)	1.47	3.91

4.4 Comparative Study of Multiplier Variants

Multiplier 1 consumed 91.5 mW of power with a latency of 3.2 ns, achieving a PSNR of 39.8 dB and an SSIM of 0.94. Multiplier 2 exhibited slightly lower power consumption at 88.3 mW and reduced latency to 2.9 ns, though its PSNR and SSIM dropped to 38.9 dB and 0.92, respectively. Multiplier 3 operated at 90.1 mW with a latency of 2.8 ns, delivering improved image quality metrics with a PSNR of 41.1 dB and an SSIM of 0.96. Multiplier 4 demonstrated the best overall performance, consuming only 86.9 mW, achieving the lowest latency of 2.7 ns, and attaining the highest image quality with a PSNR of 42.3 dB and an SSIM of 0.98. As summarized in Table 5, Multiplier 4 provided the most optimal balance between power efficiency, speed, and output quality.

Table 5: Performance Comparison of 4 Multiplier Cases

Multiplier Case	Power (mW)	Latency (ns)	PSNR (dB)	SSIM
Multiplier 1	91.5	3.2	39.8	0.94
Multiplier 2	88.3	2.9	38.9	0.92
Multiplier 3	90.1	2.8	41.1	0.96
Multiplier 4	86.9	2.7	42.3	0.98

4.5 Compressor Utilization in Dadda Tree

The use of compressors in each of the four multiplier configurations is described in Table 6. Because Design-2's logic was simpler, it used fewer 4:2 compressors than Design-1, which relied heavily on them. Accuracy was increased while resource usage was balanced with mixed designs (Multiplier 3 and 4). Multiplier 4 validated its design for resource-constrained systems by reducing component usage while preserving structural efficiency.

Table 6: Component Utilization in Dadda Tree

Component	Multiplier 1	Multiplier 2	Multiplier 3	Multiplier 4
Half Adders	3	6	4	5

3:2 Compressors	3	1	2	2
4:2 Compressors	18	17	18	17

4.6 Power Reduction under Approximation Levels

The data in Table 7 highlights how varying truncation levels influenced both error tolerance and power savings in the proposed design. At a low truncation level, the architecture maintained an error tolerance of 0.4% while achieving a 12.5% reduction in power consumption. Increasing the truncation to a moderate level raised the error tolerance to 1.2% and improved the power savings to 19.8%. The high truncation level delivered the greatest power reduction of 27.4%, though it also resulted in the highest error tolerance at 2.5%. In contrast, the adaptive truncation mode balanced performance and efficiency, maintaining an error tolerance of 1.0% while reducing power consumption by 23.1%.

Table 7: Approximation vs Power Saving

Truncation Level	Error Tolerance (%)	Power Reduction (%)
Low	0.4	12.5
Moderate	1.2	19.8
High	2.5	27.4
Adaptive	1.0	23.1

4.7 Real-Time Video Frame Performance

According to Table 8, the proposed architecture exhibited varying performance across different real-time video frame types. For low-motion frames, the system achieved the lowest frame latency of 2.1 ms, with power consumption of 89.1 mW and an entropy score of 3.2. High-motion frames slightly increased the latency to 2.4 ms and power to 91.5 mW, with a higher entropy score of 4.8. Scene change frames resulted in a latency of 2.7 ms, power consumption of 93.7 mW, and an entropy score of 5.1. Noisy input frames recorded a latency of 2.6 ms, power usage of 94.2 mW, and the highest entropy score of 5.4. These results indicate that while power and latency increased with motion complexity and noise, the system maintained stable real-time processing capabilities across all tested scenarios.

Table 8: Real-Time Video Processing Metrics

Video Frame Type	Frame Latency (ms)	Power (mW)	Entropy Score
Low-motion	2.1	89.1	3.2
High-motion	2.4	91.5	4.8
Scene Change	2.7	93.7	5.1
Noisy Input	2.6	94.2	5.4

5. Conclusion

The comprehensive analysis and hardware implementation of the proposed Additive Multiplication Module (AMM) have validated its superior performance and architectural efficiency for real-time video compression applications. Leveraging a novel combination of multiplexer-based 3:2 and 4:2 compressors, selective truncation strategies, and Dadda tree-based reduction logic, the AMM achieved significant improvements across all major performance parameters when compared to conventional full-precision multipliers. FPGA synthesis results revealed notable reductions in area utilization, latency, and power consumption, with the AMM operating at a higher clock frequency and lower critical path delay. Moreover, the incorporation of approximate arithmetic and adaptive truncation enabled a favorable trade-off between computational accuracy and energy efficiency. Error-resilient video metrics, such as PSNR (42.3 dB), SSIM (0.98), and reduced BER, MAE, and MSE values, further confirmed the architectural robustness and signal fidelity of the design. Among the four approximate multiplier cases evaluated, Multiplier 4 demonstrated the best balance of hardware efficiency and output quality, highlighting the advantage of combining exact and approximate logic structures. The adaptability of the AMM under dynamic input entropy and its seamless

integration with real-time video pipelines showcase its potential for high-throughput, low-power embedded multimedia systems. Overall, the findings support the strategic application of approximate computing in digital signal processing domains, particularly for video-centric and error-tolerant systems, while setting the groundwork for future explorations in scalable and context-aware arithmetic hardware architectures.

Abbreviation

FPGA – Field-Programmable Gate Array; AMM – Additive Multiplication Module; HDL – Hardware Description Language; PSNR – Peak Signal-to-Noise Ratio; SSIM – Structural Similarity Index; CR – Compression Ratio; BER – Bit Error Rate; MAE – Mean Absolute Error; MSE – Mean Squared Error; VVC – Versatile Video Coding; DCT – Discrete Cosine Transform; WMSNs – Wireless Multimedia Sensor Networks; DSP – Digital Signal Processing; LUT – Look-Up Table; CLB – Configurable Logic Block; FSM – Finite State Machine; RTL – Register Transfer Level; and AI – Artificial Intelligence.

References

1. Frustaci, F., Perri, S., Corsonello, P., & Alioto, M. (2020). Approximate multipliers with dynamic truncation for energy reduction via graceful quality degradation. IEEE Transactions on Circuits and Systems II: Express Briefs, 67(12), 3427–3431.
2. Saif, S., Abbas, H. M., Nassar, S. M., & Wahdan, A. A. (2007). An FPGA implementation of a neural optimization of block truncation coding for image/video compression. Microprocessors and Microsystems, 31(8), 477–486.
3. Deepa, M., Reddy, D. A., Hariprasath, K., Rohit, N., & Yedusheker, S. (2024). Design of efficient approximate unsigned multiplier using various approximate compressor configurations. Proceedings of the IEEE International Conference on Smart Systems, Electrical, Electronics, Communication and Computer Engineering (ICSSEECC), 415–420.
4. Wang, Y., Wang, Y., Li, H., & Li, X. (2021). An efficient deep learning accelerator architecture for compressed video analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(9), 2808–2820.
5. Schwarz, H., Coban, M., Karczewicz, M., Chuang, T. D., Bossen, F., Alshin, A., Lainema, J., Helmrich, C. R., & Wiegand, T. (2021). Quantization and entropy coding in the versatile video coding (VVC) standard. IEEE Transactions on Circuits and Systems for Video Technology, 31(10), 3891–3906.
6. Strollo, A. G. M., De Caro, D., Napoli, E., Petra, N., & Di Meo, G. (2020). Low-power approximate multiplier with error recovery using a new approximate 4-2 compressor. Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 1–4.
7. Wan, S., Yang, F., & Izquierdo, E. (2009). Lagrange multiplier selection in wavelet-based scalable video coding for quality scalability. Signal Processing: Image Communication, 24(9), 730–739.
8. Wan, S., Yang, F., & Izquierdo, E. (2009). Lagrange multiplier selection in wavelet-based scalable video coding for quality scalability. Signal Processing: Image Communication, 24(9), 730–739.
9. Yang, Z., Li, X., & Yang, J. (2020). Approximate compressor-based multiplier design methodology for error-resilient digital signal processing. Journal of Circuits, Systems and Computers, 29(14), 2050233.
10. Li, M. C., Ghosh, A., & Sen, S. (2024). Approximate DCT and quantization techniques for energy-constrained image sensors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Early Access).
11. Kumar, U. A., Jain, N., Chatterjee, S. K., & Ahmed, S. E. (2020). Evaluation of multiplier-less DCT transform using in-exact computing. Proceedings of the International Conference on Machine Learning, Image Processing, Network Security and Data Science, 11–23.
12. Nesam, J. J. J., & Sivanantham, S. (2020). Efficient half-precision floating point multiplier targeting color space conversion. Multimedia Tools and Applications, 79(1), 89–117.
13. Nirmala, R., Begum, S. A., Selvanayagi, A., & Ramya, P. (2024). VLSI implementation of an energy-efficient color image compressor using improved block truncation coding and enhanced Golomb–Rice coding for wireless sensor networks. The Journal of Supercomputing, 80(17), 24807–24834.
14. Banerjee, R., & Das Bit, S. (2019). Low-overhead video compression combining partial discrete cosine transform and compressed sensing in WMSNs. Wireless Networks, 25(8), 5113–5135.
15. Lotrič, U., Pilipović, R., & Bulić, P. (2021). A hybrid radix-4 and approximate logarithmic multiplier for energy efficient image processing. Electronics, 10(10), 1175.
16. Strollo, A. G. M., Napoli, E., De Caro, D., Petra, N., & Di Meo, G. (2020). Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers. IEEE Transactions on Circuits and Systems I: Regular Papers, 67(9), 3021–3034.
17. Faraji, S. R., Abillama, P., & Bazargan, K. (2020). Low-cost approximate constant coefficient hybrid binary-unary multiplier for DSP applications. Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 93–101.
18. Faraji, S. R., & Bazargan, K. (2020). Hybrid binary-unary truncated multiplication for DSP applications on FPGAs. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1–9.